



ver1.0 – April 29, 2003

## CEUSB2 GUI SPECIFICATIONS

CeUsb2 Graphical User Interface  
Documentation for Software Programmers

## CONTENTS

<b>CONTENTS .....</b>	<b>1</b>
<b>1 INTRODUCTION.....</b>	<b>2</b>
1.1 About This Documentation.....	2
1.2 Prerequisites .....	2
1.3 Distribution .....	2
1.4 Overview .....	3
<b>2 USING CEUSB2 GUI.....</b>	<b>3</b>
2.1 Enumerating Devices.....	3
2.2 Displaying Device Properties .....	4
2.3 Descriptor Dialog .....	5
2.4 Handling USB Configurations .....	6
2.5 Handling USB Interfaces and Alternate Settings.....	7
2.6 USB Vendor or Class Requests.....	8
2.7 Feature Requests .....	10
2.8 Controlling the FX2 CPU.....	11
2.9 Controlling the GPIF .....	12
2.10 RS232 - Serial Input Output.....	14
2.11 Controlling the FPGA .....	15
2.12 Error Handling.....	16
<b>3 CEUSB2 GUI REFERENCE .....</b>	<b>17</b>
3.1 Dialog Functions .....	18
3.1.1 EnumDevDlg.....	18
3.1.2 DevPropertiesDlg.....	18
3.1.3 DescriptorsDlg .....	19
3.1.4 SelectConfigDlg .....	19
3.1.5 UsbInterfaceDlg .....	19
3.1.6 VendorOrClassReqDlg .....	20
3.1.7 FeatureReqDlg .....	20
3.1.8 CpuCtrlDlg .....	20
3.1.9 GpifCtrlDlg .....	21
3.1.10 RS232Dlg .....	21
3.1.11 FpgaCtrlDlg.....	22
3.1.12 ConfigureFpgaDlg.....	22
3.2 Error Box Functions .....	23
3.2.1 ErrorBox.....	23
3.2.2 ErrorBoxEx .....	23

3.3	Exported Helper Functions.....	24
3.3.1	WsSetDlgItemText.....	24
3.3.2	OpenSaveFileDialog.....	25

## 1 INTRODUCTION

### 1.1 About This Documentation

CeUsb2 GUI is the graphical user programming interface for CeUsb2 compatible devices from CESYS GmbH. It has several exported global functions which open some Windows dialogs for general USB functionality, CPU and GPIF configuration, FPGA configuration, error handling and other features of CeUsb2 devices. Before reading this documentation it would be helpful to check general CeUsb2 design guide documentation, CeUsb2dg.pdf, to understand the components which comprise the complete CeUsb2 software, and run together with the GUI executable.

### 1.2 Prerequisites

An intermediate or upper level knowledge of C or C++ language is necessary to understand the GUI functions and programming examples given throughout this documentation.

CeUsb2 boards are USB2.0 devices, so it is necessary to understand basics of the USB protocol in order to understand the USB communication and data transfer operations.

### 1.3 Distribution

After you install CeUsb2 software, you can find the header file for the CeUsb2 GUI, CeUsb2Wr.h, in CeUsb2\inc directory. It includes all function definitions of the GUI. You will also need guides.h header file for the constant GUID definitions of the current CeUsb2 device interfaces.

GUI executable, CeUsb2Mfc.dll, is a Win 32 dynamic link library (DLL). Although it is written using the MFC (Microsoft Foundation Classes), it can be used by a non MFC application. Therefore, you should be able to link this DLL with any 32-bit Windows based project written in C++. CeUsb2Mfc.lib library file is also required for the linker. Both the DLL and the library file can be found in lib\fre\i386 directory (debug versions are in lib\chk\i386).

## **1.4 Overview**

CeUsb2 graphical user programming interface (GUI) stands on the CeUsb2 application programming interface (API), and uses its classes and functions. For more information about the CeUsb2 API check CeUsb2 API specifications document, CeUsb2Api.pdf.

It is possible for the programmers to use API only and write a complete project in C++ language. However the GUI simplifies the usage of the API by implementing upper level functions for opening some device control dialogs and for more sophisticated error handling.

Chapter 2 explains the usage of the GUI with some example codes. All of these codes are tested in separate projects and most of them are ready to be used as templates.

Chapter 3 is a reference for the GUI. It explains the GUI functions with their arguments and return values.

## **2 USING CEUSB2 GUI**

### **2.1 Enumerating Devices**

Almost all functions of the CeUsb2 GUI require a pointer to an instance of CeUsb2 device handler class. The required operation is performed over this device instance. Therefore, available CeUsb2 devices in the system should be searched and the target device should be opened before any other operations are performed.

EnumDevDlg function of the GUI opens a Windows dialog for device enumeration as seen in figure 2.1. It can enumerate the CeUsb2 devices in the system with a given 128 GUID number identifying the CeUsb2 driver interface and a pointer to an instance of CeUsb2 class.

After the enumeration, it displays the CeUsb2 devices found in a list box; from which the user can select one for further operation, by double clicking on its name or by clicking on the OK button after selecting the right entry in the list. If a device is selected successfully, its previous device instance (if exists) will be closed, the new devices features will be copied and the new device will be opened. If CANCEL button is pressed; the dialog exits without making any changes.

The following code shows how to open the device enumeration dialog. The resultant dialog can be seen in figure 2.1.

```
// Code example 2.1
// Device enumeration dialog

#include <stdio.h>
#include <windows.h>
#include <tchar.h>

#include <Initguid.h> // necessary for GUID macros
#include "..\..\inc\CeUsb2Wr.h" // GUI main include header file
#include "..\..\inc\guids.h" // includes CeUsb2 device interface GUID

extern HWND ghWnd ; // handle to the owner window
CeUsb2 Dev; // instance of device handler class
int ExitCode; // dialog exit code

// open enumerate devices dialog
ExitCode = EnumDevDlg(
    GUID_INTERFACE_CEUSB2,
    ghWnd,
    FALSE,
    &Dev
);
printf("Dialog exited with code %d\n", ExitCode);
```

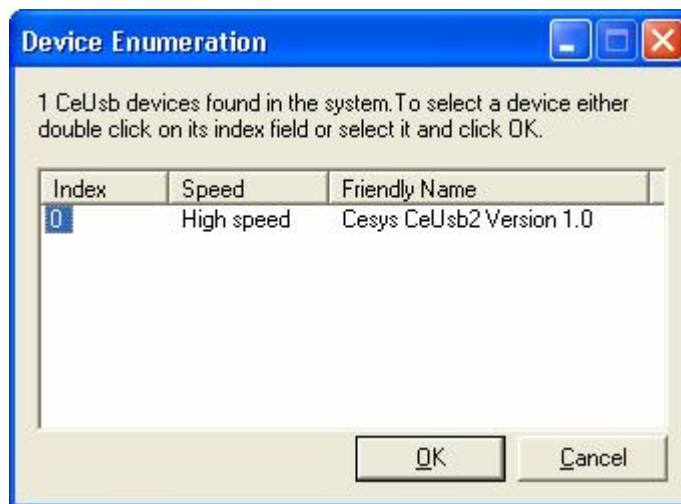


Figure 2.1 – Device Enumeration Dialog

## 2.2 Displaying Device Properties

DevPropertiesDlg function of the GUI opens a Windows dialog with versioning information for the CeUsb2 driver, current firmware, application programming

interface (API) and the GUI. Code example 2.2 shows how to open device properties dialog and the resultant dialog can be seen in figure 2.2.

```
// Code example 2.2
// Device properties dialog

ExitCode = DevPropertiesDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

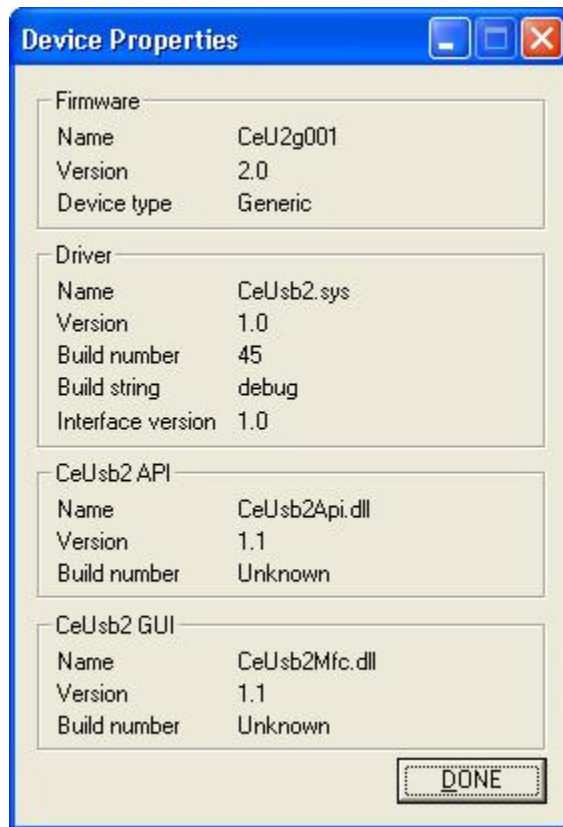


Figure 2.2 – Device Properties Dialog

## 2.3 Descriptor Dialog

Like all other USB devices CeUsb2 devices report their features and capabilities with USB descriptors. DescriptorsDlg function of the GUI opens a Windows dialog with a tree view displaying the descriptors which is defined by the current firmware application running on the board. If an entry in the tree view is double clicked, the corresponding USB descriptor's fields are displayed in the static text field below.

Currently descriptors dialog can display device, configuration, interface, endpoint and string descriptors.

Code example 2.3 shows how to open device descriptors dialog and the resultant dialog can be seen in figure 2.3.

```
// Code example 2.3
// USB descriptors dialog

ExitCode = DescriptorsDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

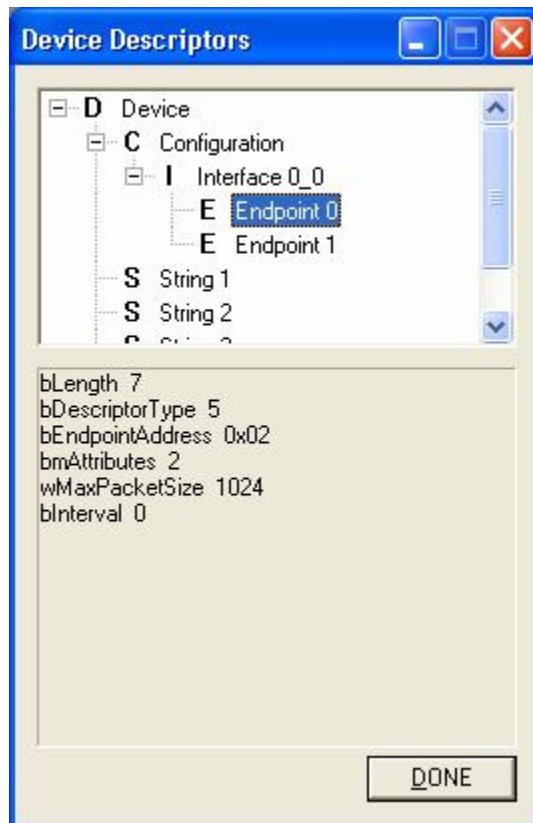


Figure 2.3 – USB Descriptors Dialog

## 2.4 Handling USB Configurations

Some USB devices have more than one USB configurations each of which defines another behavior of the device. The user is able to change the current USB configuration dynamically. Currently all CeUsb2 firmwares define only one configuration. However some firmwares in the future might define more than one USB configurations. Therefore CeUsb2 GUI defines the function `SelectConfigDlg` which opens a dialog with the available USB configurations defined in the device, in a drop down combo box. The user can change the configuration number in the combo box and change it in the device by pressing the `Select Configuration`

button. Notice that USB configurations are indexed starting from 1. A configuration with the index 0 indicates an unconfigured USB state. DONE button simply closes the dialog without making any changes.

Code example 2.4 shows how to open select configuration dialog, and the resultant dialog can be seen in figure 2.4.

```
// Code example 2.4
// Select USB configuration dialog

ExitCode = SelectConfigDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

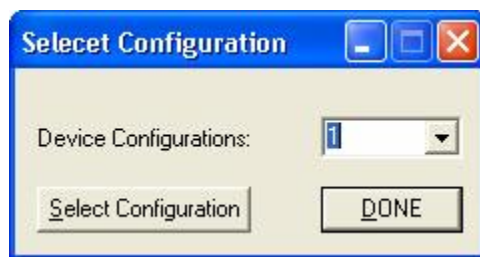


Figure 2.4 – Select USB Configuration Dialog

## 2.5 Handling USB Interfaces and Alternate Settings

Like USB configurations, current USB interface and alternate setting can be changed with the CeUsb2 software. `UsbInterfaceDlg` function is used for this purpose. It opens a dialog with information about the current interface – alternate setting pair (figure 2.5).

Upper part of the dialog displays the current interface number and alternate setting number in two different drop-down combo boxes. The user can select different values from the combo boxes and change the maximum transfer sizes of the USB communication pipes, defined in the interface, from the edit boxes in the right part of the dialog. Select Interface button re-selects an USB interface with the user's changes.

Bottom part of the dialog displays the pipe information for the currently selected interface. Target pipe can be changed from the Pipe Number combo box. The user can reset and abort a pipe with Reset Pipe and Abort Pipe buttons. DONE button simply closes the dialog without making any changes.

Code example 2.5 shows how to open select interface dialog, and the resultant dialog can be seen in figure 2.5.

```
// Code example 2.5
// Select USB configuration dialog
```

```
ExitCode = UsbInterfaceDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

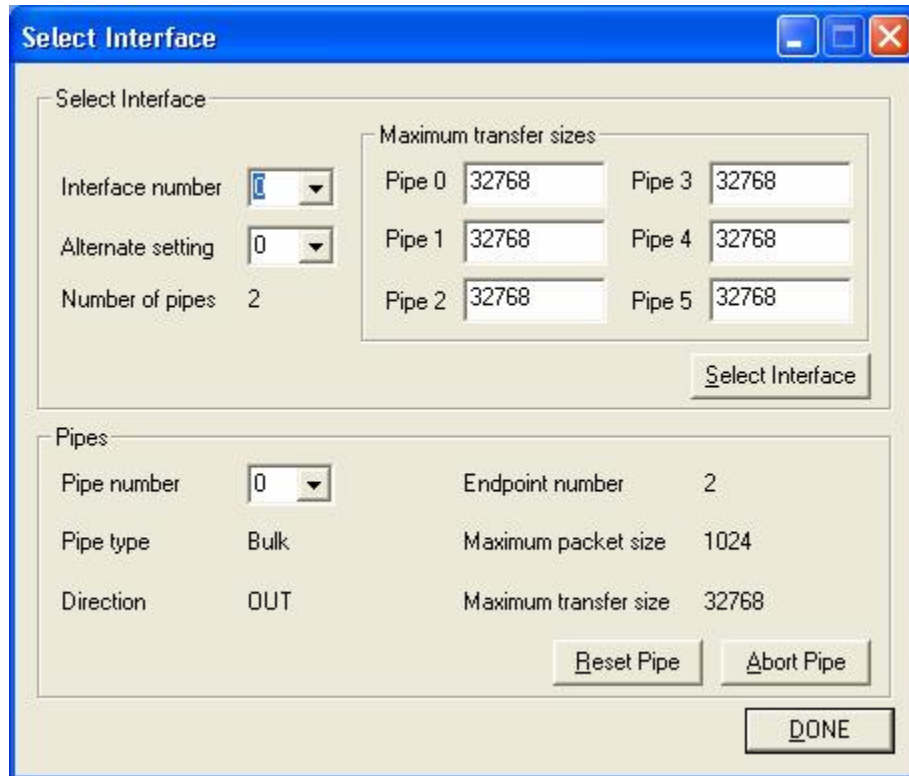


Figure 2.5 – Select USB Interface Dialog

## 2.6 USB Vendor or Class Requests

VendorOrClassReqDlg function of the GUI opens a dialog on which the user can perform vendor or class specific CeUsb2 requests. Upper left part of the dialog has three radio buttons which select the type of the operation. Upper right part has also three radio buttons which select the request recipient. Vendor or class requests' recipient can be an USB device, interface, endpoint or another device defined target (this last option is not implemented on this dialog).

Bottom part of the dialog selects the request which will be directed to the device.

Some predefined vendor requests can be selected form the Request combo box. Request code text box is the hexadecimal code of the request (1 byte).

Direction radio buttons select the direction of the request (OUT = from host to device, IN = from device to the host).

Value is the request specific 2 byte value in hexadecimal format.

Index is the request specific 2 byte index in hexadecimal format.

Data Length is the length of the data buffer which will be transferred with the request.

Data text field displays the data received from the device or data that will be send to the device in hexadecimal format. If you perform an OUT request, write your data bytes in two character hexadecimal format with spaces between them. For example: 00 AB 02 20 C DD ....

Bytes Transferred static text field displays the actual amount of data transferred after the request is completed.

Execute button executes the vendor or class request with the current settings of the dialog. DONE button closes the dialog without performing any operations on the device.

Code example 2.6 shows how to open vendor or class request dialog, and the resultant dialog can be seen in figure 2.6.

```
// Code example 2.6
// Select USB interface dialog

ExitCode = VendorOrClassReqDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

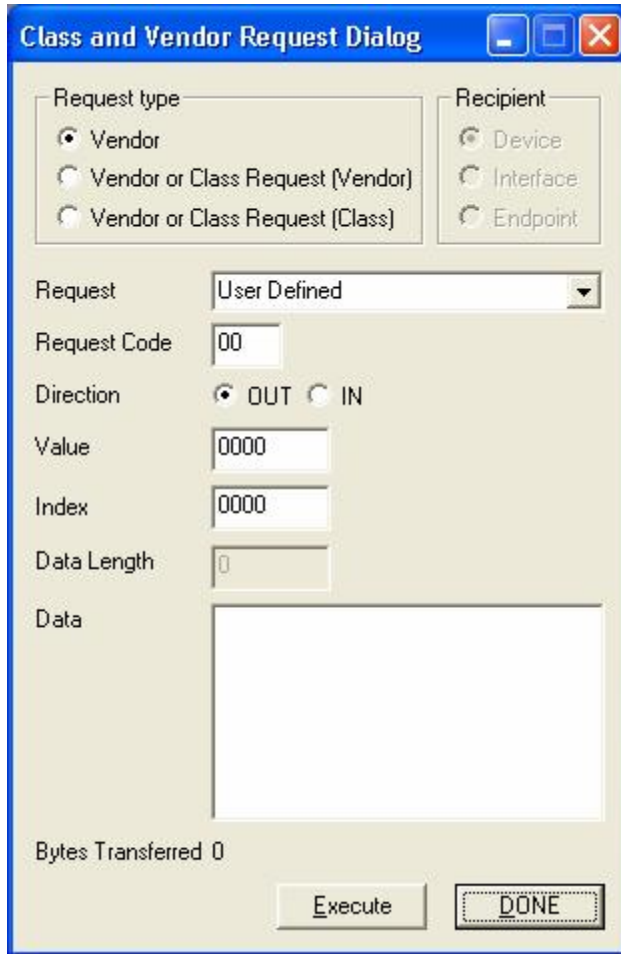


Figure 2.6 – USB Vendor or Class Request Dialog

## 2.7 Feature Requests

FeatureReqDlg function of the GUI opens a Windows dialog on which the user can perform USB set and clear feature requests through CeUsb2 devices. On this dialog, Recipient combo box selects the recipient for the request. Feature requests' recipient can be an USB device, interface, endpoint or another device defined target (this last option is not implemented on this dialog).

Feature Selector text field is the 2 byte hexadecimal feature code.  
Index text field is the 2 byte hexadecimal index value specific for the current USB feature.

The user can select and clear the selected feature with Select Feature and Clear Feature buttons. DONE button simply closes the dialog without making any changes.

Code example 2.7 shows how to open feature request dialog, and the resultant dialog can be seen in figure 2.7.

```
// Code example 2.7
// USB feature request dialog

ExitCode = FeatureReqDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```



Figure 2.7 – USB Feature Request Dialog

## 2.8 Controlling the FX2 CPU

CpuCtrlDlg function of the GUI opens the CPU Control Dialog on which the user can change some settings of the FX2 CPU (enhanced 8051 microprocessor).

CPU speed radio buttons select the clock speed of the CPU (12, 24 or 48 MHz). If CPU clock inverted check box is checked, CPU clock is inverted. If CPU clock output enable check box is checked, CPU clock is given out from the FX2 chip on a pin (check your board's hardware documentation to see which pin is the output clock).

Changes made by the user will not take affect until the Re-initialize button is pressed. It forces the firmware, to jump to its initialization function. Soft Reset button forces the firmware to perform a soft reset (vector to org 00h). The changes for the CPU is lost if this button is pressed. DONE button simply closes the dialog without making any changes.

Code example 2.8 shows how to open CPU control dialog, and the resultant dialog can be seen in figure 2.8.

```
// Code example 2.8
// CPU control dialog

ExitCode = CpuCtrlDlg(&Dev,ghWnd);
```

```
printf("Dialog exited with code %d\n", ExitCode);
```



Figure 2.8 – CPU Control Dialog

## 2.9 Controlling the GPIF

Most of the CeUsb2 devices perform high bandwidth data transfers with the GPIF (see CeUsb2 design guide for more information about GPIF). For such devices, the GUI serves the function `GpifCtrlDlg`, which opens a Windows dialog for controlling the GPIF engine. Lower left part of the dialog controls the FX2 CPU and doubles the functionality of the CPU control dialog (see 2.8, controlling the FX2 CPU). Upper left part has the fields for GPIF initialization settings:

Interface clock source radio buttons select the source of the GPIF interface clock (IFCLK on circuit diagrams). GPIF can use 30 or 48 MHz internal clock or 5 to 48 MHz external clock.

Interface clock speed radio buttons select the GPIF interface clock speed (30 or 48 MHz). If the clock source is selected as external, this field doesn't have any effects.

GPIF Operation Mode radio buttons selects between the auto and manual modes of the GPIF engine. Check your firmware's documentation to see which mode(s) are supported.

If Interface Clock Output Enable check box is checked, interface clock is also given out on a pin from the FX2 chip (check your board's hardware documentation to see which pin is the output interface clock - IFCLK).

If Invert Clock Polarity check box is checked, interface clock is inverted.

If Word-wide (16 bit data interface) check box is checked, GPIF uses 16 bit data interface (16 data pins); otherwise it uses 8 bit data interface.

GPIF engine can be programmed with a initialization array and waveform data so that its signaling mechanism is determined for different applications. This data can be updated from the right part of the GPIF control dialog.

7 byte GPIF initialization data is divided into 7 edit boxes. These fields are one byte hexadecimal values.

128 byte GPIF waveform data can be retrieved or updated from the large scrolling text field. This data is in hexadecimal format with spaces between two preceding bytes.

Load button loads GPIF initialization data and waveform data from a text file (with extension txt) to the edit boxes.

Save button saves GPIF initialization data and waveform data from the edit boxes to a text file (with extension txt).

GPIF data text file format is as follows:

Init Data = D0 D1 D2 D3 D4 D5 D6 D7

Waveform Data = W0 W1 ..... W127

New line and tab characters can be also used inside the text file.

All changes made will not take effect until the Refresh button is pressed. It reinitializes the firmware with the new settings. Restore button changes the setting and waveform data to their initial states (initial states are stored when the dialog is opened).

DONE button simply closes the dialog without making any changes.

Code example 2.9 shows how to open GPIF control dialog, and the resultant dialog can be seen in figure 2.9.

```
// Code example 2.9
// GPIF Control dialog

ExitCode = GpifCtrlDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

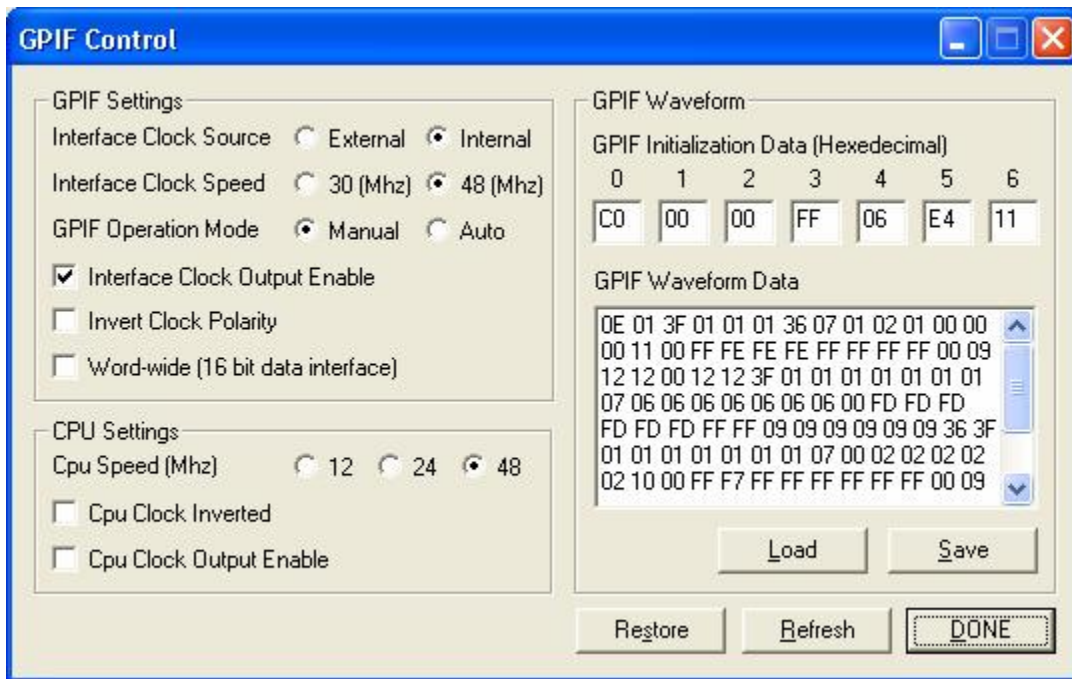


Figure 2.9 – GPIF Control Dialog

## 2.10 RS232 - Serial Input Output

RS232Dlg function of the GUI opens a dialog on which the user can send and receive data over serial interface of the CeUsb2 device, if it has at least one serial interface (check your board's hardware documentation to see if it supports serial interface I/O).

Users can enter a text in the Transmit Buffer text field of the Rs232 dialog. This text is sent over serial line when Send button is pressed.

Start Rx (Stop Rx) button starts and stops data capturing over the serial line. Incoming serial data (if available) can be seen in the Receive Buffer text field. Clear button clears the contents of the Receive Buffer text field. DONE button simply closes the dialog without making any changes.

Code example 2.10 shows how to open RS232 – Serial I/O dialog, and the resultant dialog can be seen in figure 2.10.

```
// Code example 2.10
// RS232 - Serial interface dialog

ExitCode = RS232Dlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```

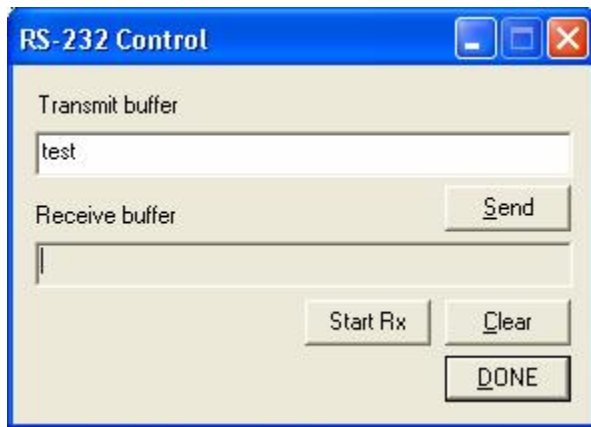


Figure 2.10 – RS232 Serial I/O Dialog

## 2.11 Controlling the FPGA

Some CeUsb2 type boards have an on-board FPGA chip. FpgaCtrlDlg function of the GUI opens a dialog on which the user can control this FPGA chip.

With the FPGA Control Dialog, the user can select an FPGA configuration file (with extensions exo or rbt) with Select>> button. It opens a File Open Dialog in which the user can select the file name. If a valid file is selected, its path is displayed in the upper text field.

In the middle part of the dialog there are some controls which change some settings for FPGA configuration.

If Use Control Endpoint check box is selected, FPGA configuration will be performed over the default USB control endpoint (endpoint 0). Check your firmware documentation to see which endpoint used for this operation.

If Search for a valid pipe check box is checked, a suitable USB pipe (endpoint) will be searched automatically by the software. If it is not checked, the number of the desired pipe can be entered to the Pipe Number text field. A pipe number of -1 also indicates auto searching for a suitable pipe.

If the Transfer package size auto check box is checked, the optimum package size is detected by the software automatically. Alternatively, the user can give a package size in the right edit box.

Configure button configures the FPGA with the given configuration file path and selected settings.

Clear button erases the current FPGA configuration from the FPGA chip.

Reset button sends a reset signal to the currently running FPGA design.  
DONE button simply closes the dialog without making any changes.

Another way of configuring the FPGA is using the ConfigureFpgaDlg of the GUI.  
Check GUI reference section to learn more about this function.

Code example 2.11 shows how to open FPGA control dialog, and the resultant dialog can be seen in figure 2.11.

```
// Code example 2.11
// FPGA control dialog

ExitCode = FpgaCtrlDlg(&Dev,ghWnd);
printf("Dialog exited with code %d\n", ExitCode);
```



Figure 2.11 – FPGA Control Dialog

## 2.12 Error Handling

ErrorBox and ErrorBoxEx functions of the GUI open windows message boxes with a short description of an error with the given error code.

ErrorBox function displays a user supplied text on the top of the message box. Below, it displays the 32 bit error code and a description of the error, either supplied by CeUsb2 API or Windows system.

ErrorBoxEx function is similar to MessageBox except that it can display also another description text with the status of the device, if a USB specific error is occurred. Moreover, it can open a message box, with OK and CANCEL buttons.

Code example 2.12 shows the simple usage of the ErrorBox function. The resultant error message box is shown in figure 2.12.

```
// Code example 2.12
// Error message box

ULONG PipeNumber = 1;

// sample function from the CeUsb2 API
DWORD dwStatus = Dev.ResetPipe(PipeNumber);
if(!CE_SUCCESS(dwStatus))
{
    // error!!!
    // report it to the user
    ErrorBox(
        ghWnd,          // handle to owner window
        dwStatus,       // error code
        TRUE,           // search the reason for the error
        _T("Unable to reset pipe %d"), // user description
        PipeNumber      // format argument
    );
}
```



Figure 2.12 – Sample Error Message Box

### 3 CEUSB2 GUI REFERENCE

This section is a reference to the exported functions of the CeUsb2 GUI. Most of these functions open a modal Windows dialog. When the dialog is closed by the user or because of an internal error, then the function returns with the exit code of the dialog. Table 3.1 explains the possible exit codes.

Exit Code	MFC definition	Description
-----------	----------------	-------------

-1	None	Reported by Win32, means the dialog box could not be created.
3	IDABORT	Reported by Win32, means that an error occurred when creating the dialog box.
2	IDCANCEL	Reported by the GUI when Cancel button is pressed or an error is detected which requires the termination of the dialog.
1	IDOK	Reported by the GUI when OK button is pressed, or execution of the dialog is finished successfully.

Table 3.1 – GUI dialog functions' return codes.

If you use the GUI from a MFC (Microsoft foundation classes) project, then you can use the MFC definitions, given in table 3.1, directly.

## 3.1 Dialog Functions

### 3.1.1 EnumDevDlg

```
int EnumDevDlg(
    const GUID    &Guid,
    HWND         hwParent,
    BOOL         bHideIfOnlyOnePresent,
    CeUsb2      *pDevice
);
```

**Description** : Enumerates CeUsb2 devices in the system. Upon request, displays a dialog with a list of the devices found. The user can choose one for further operation.

**Return Value** : Exit code of the dialog.

**Arguments** :

Guid - 128 bit CeUsb2 interface GUID number.

hwParent - Handle to the window which owns this dialog.

bHideIfOnlyOnePresent - If TRUE, and if only one device is found, then the dialog box is hidden, and the device found is selected for further operation.

pDevice - Pointer to an instance of device handler class, CeUsb2.

### 3.1.2 DevPropertiesDlg

```
int DevPropertiesDlg(
    CeUsb2      *pDevice,
    HWND         hwParent
```

```
);
```

**Description** : Displays a dialog with versioning information of the CeUsb2 driver, firmware API and GUI.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.3 DescriptorsDlg

```
int DescriptorsDlg (  
    CeUsb2      *pDevice,  
    HWND       hwParent  
);
```

**Description** : Displays a dialog with the USB descriptors of the CeUsb2 device and all their fields.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.4 SelectConfigDlg

```
int SelectConfigDlg(  
    CeUsb2      *pDevice,  
    HWND       hwParent  
);
```

**Description** : Displays a dialog box with the current USB configuration number of the CeUsb2 device. The user can also change the configuration or put the device in an unconfigured state.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.5 UsbInterfaceDlg

```
int UsbInterfaceDlg (  
    CeUsb2      *pDevice,
```

```
HWND    hwParent
);
```

**Description** : Displays a dialog box with the current USB interface and alternate setting of the CeUsb2 device together with current pipe information. User can change the maximum transfer sizes of the pipes.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.6 VendorOrClassReqDlg

```
int VendorOrClassReqDlg (
    CeUsb2    *pDevice,
    HWND      hwParent
);
```

**Description** : Displays a dialog which can be used to perform USB vendor or class requests through CeUsb2 devices.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.7 FeatureReqDlg

```
int FeatureReqDlg (
    CeUsb2    *pDevice,
    HWND      hwParent
);
```

**Description** : Displays a dialog which can be used to perform USB feature requests through CeUsb2 devices.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.8 CpuCtrlDlg

```
int CpuCtrlDlg (  
    CeUsb2      *pDevice,  
    HWND       hwParent  
);
```

**Description** : Displays a dialog which can be used to change the FX2 CPU settings.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.9 GpifCtrlDlg

```
int GpifCtrlDlg (  
    CeUsb2      *pDevice,  
    HWND       hwParent  
);
```

**Description** : Displays a dialog which can be used for changing the GPIF settings and for retrieving & loading GPIF Initialization and Waveform data.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.10 RS232Dlg

```
int RS232Dlg (  
    CeUsb2      *pDevice,  
    HWND       hwParent  
);
```

**Description** : Displays a dialog which can be used to make data transfer over RS232 interface of the CeUsb2 device.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.11 FpgaCtrlDlg

```
int FpgaCtrlDlg (  
    CeUsb2      *pDevice,  
    HWND       hwParent  
);
```

**Description** : Displays a dialog which can be used to control and configure the FPGA. For a quick FPGA configuration use ConfigureFpgaDlg function instead.

**Return Value** : Exit code of the dialog.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

hwParent - Handle to the window which owns this dialog.

### 3.1.12 ConfigureFpgaDlg

```
DWORD ConfigureFpgaDlg(  
    CeUsb2      *pDevice,  
    LPTSTR      lpConfigFile,  
    LPCTSTR     lpInitialDirectory,  
    HWND       hwParent,  
    BOOL        bControlTransfer,  
    int         PipeNum,  
    BOOL        bUseDynamicGpifWvf,  
    ULONG       PackageSize  
);
```

**Description** : Configures the FPGA with a given configuration file path. If the given file path is not found, it opens an open file dialog on which the user can select another configuration file.

**Return Value** : 32 bit status code if an error occurs, else 0.

**Arguments** :

pDevice - Pointer to an instance of device handler class, CeUsb2.

lpConfigFile - Configuration file path (with extension .exo or .rbt).

lpInitialDirectory - Initial configuration file search directory.

hwParent - Handle to the window which owns this dialog

bControlTransfer – Boolean variable. TRUE = configuration is handled over USB control pipe, FALSE = configuration is handled over an USB bulk or isochronous pipe.

PipeNum - Pipe number on which configuration data is sent to the Fpga device.

A pipe number of -1 indicates auto searching of a suitable pipe for FPGA configuration. If bControlTransfer is TRUE, this argument doesn't have any affect.

bUseDynamicGpifWvf – Boolean variable. If TRUE, internally stored GPIF waveform is used dynamically for FPGA configuration. Previous waveform is

restored after configuration is done. If bControlTransfer is TRUE, this argument doesn't have any affect.

PackageSize - USB data transfer packet size. Use 0xFFFFFFFF for automatic size detection. This argument is used for the firmwares which use bulk pipes to configure the FPGA.

## 3.2 Error Box Functions

### 3.2.1 ErrorBox

```
void ErrorBox(  
    HWND      hWnd,  
    DWORD     dwError,  
    BOOL      bSearchReason,  
    LPCTSTR   szUserDescription,  
    ...  
);
```

**Description** : Displays a message box with a description of an error.

**Return Value** : None

**Arguments** :

hWnd - Handle of the window which will own this message box.

dwError - Error status code.

bSearchReason - If TRUE, the reason for the error is searched first in the API and the driver, then in the system; if not only the user description string is displayed.

szUserDescription - User description format string (First line of the message box).

... - User description format arguments.

### 3.2.2 ErrorBoxEx

```
int ErrorBoxEx(  
    HWND      hWnd,  
    DWORD     dwError,  
    BOOL      bSearchReason,  
    BOOL      bCheckUSBDError,  
    BOOL      bOkCancel,  
    CeUsb2    *pDevice,  
    LPCTSTR   szUserDescription,  
    ...  
);
```

```
);
```

**Description** : Displays a message box with a description of an error. It can also display USB specific error descriptions.

**Return Value** : Message box exit code.

**Arguments** :

hWnd - Handle of the window which will own this message box.

dwError - Error status code.

bSearchReason - If TRUE, the reason for the error is searched first in the API and the driver, then in the system; if not only the user description string is displayed.

bCheckUSBDError - If TRUE, searches for a possible USB D error.

bOkCancel - If TRUE, dialog box is displayed with OK and CANCEL buttons; otherwise it is displayed with an OK button only.

pDevice - Pointer to an instance of device handler class, CeUsb2. Will be used to retrieve the last occurred USB D error. Therefore, if bCheckUSBDError is not TRUE this argument will not be used.

szUserDescription - User description format string (First line of the message box).

... - User description format arguments.

### 3.3 Exported Helper Functions

#### 3.3.1 WsSetDlgItemText

```
void WsSetDlgItemText(  
    HWND      hParent,  
    UINT      uiControlId,  
    LPTSTR    lpFormat,  
    ...  
);
```

**Description** : Formats a text and displays it on a dialog item.

**Return Value** : None

**Arguments** :

hParent - Handle of the owner dialog of the item.

uiControlId - Dialog item's identifier.

lpFormat - Format string.

... - One or more optional arguments for text formatting.

### 3.3.2 OpenSaveFileDialog

```
DWORD OpenSaveFileDialog(  
    HWND        hWnd,  
    BOOL        bOpen,  
    LPCTSTR     lpInitialDir,  
    LPCTSTR     lpFilter,  
    LPTSTR      lpFileName,  
    LPCTSTR     lpDefaultExtension  
);
```

**Description** : Opens the Win32 file open & save dialog.

**Return Value** : None

**Arguments** :

hWnd - Handle to the owner window.

bOpen - If TRUE, open file dialog; else save file dialog is opened.

lpInitialDir - Initial file search directory.

lpFilter - Filter string. For example the following filter string is used to open or save text files with extension txt:

```
LPCSTR pszOpenTxtFormatStr = \  
    "Text File (*.txt)"    "\\0"  "*.txt" "\\0"  \  
    "All Files (*.*)"     "\\0"  "*.*"  "\\0"  \  
    "\\0";
```

lpFileName – Path of the selected file.

lpDefaultExtension: Default file extension (without \*. characters).